


# EFFICIENT SNOOP FILTER IN A MULTIPLE-PROCESSOR-BUS SYSTEM

By:

Paul B. Rawlins  
Phil M. Jones

EXPRESS MAIL MAILING LABEL	
NUMBER:	EL 827 072 475
DATE OF DEPOSIT:	September 28, 2001
<i>Pursuant to 37 C.F.R. § 1.10, I hereby certify that I am personally depositing this paper or fee with the U.S. Postal Service, "Express Mail Post Office to Addressee" service on the date indicated above in a sealed envelope (a) having the above-numbered Express Mail label and sufficient postage affixed, and (b) addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.</i>	
09/28/01 Date	 Signature

# EFFICIENT SNOOP FILTER IN A MULTIPLE-PROCESSOR-BUS SYSTEM

## BACKGROUND OF THE INVENTION

### 1. Field Of The Invention

This invention relates generally to a multi-processor-bus memory system and, more particularly, to an efficient mechanism for filtering processor cache snoops in a multi-processor-bus-system.

### 2. Description Of The Related Art

This section is intended to introduce the reader to various aspects of art which may be related to various aspects of the present invention which are described and/or claimed below. This discussion is believed to be helpful in providing the reader with background information to facilitate a better understanding of the various aspects of the present invention. Accordingly, it should be understood that these statements are to be read in this light, and not as admissions of prior art.

The use of computers has increased dramatically over the past few decades. In years past, computers were relatively few in number and primarily used as scientific tools. However, with the advent of standardized architectures and operating systems, computers soon became virtually indispensable tools for a wide variety of business applications. Perhaps even more significantly, in the past ten to fifteen years with the advent of relatively simple user interfaces and ever increasing processing capabilities, computers have now found their way into many homes.

The types of computer systems have similarly evolved over time. For example, early scientific computers were typically stand alone systems designed to carry out relatively specific tasks and required relatively knowledgeable users. As computer systems evolved into the business arena, mainframe computers emerged. In mainframe systems, users utilized “dumb” terminals to provide input to and to receive output from the mainframe computer while all processing was done centrally by the mainframe computer. As users desired more autonomy in their choice of computing services, personal computers evolved to provide processing capability on each users desktop. More recently, personal computers have given rise to relatively powerful computers called servers. Servers are typically multi-processor computers that couple numerous personal computers together in a network. In addition, these powerful servers are also finding applications in various other capacities, such as in the communications and Internet industries.

Computers today, such as the personal computers and servers discussed above, rely on microprocessors, associated chip sets, and memory chips to perform most of their processing functions. Because these devices are integrated circuits formed on semi-conducting substrates, the technological improvements of these devices have essentially kept pace with one another over the years. In contrast to the dramatic improvements of the processing portions of the computer system, the mass storage portion of the computer system has experienced only modest growth in speed and reliability. As a result, computer systems failed to capitalize fully on the increased speed of the improving processing systems due to the dramatically inferior capabilities of the mass data storage devices coupled to the systems.

There are a variety of different memory devices available for use in microprocessor-based systems. The type of memory device chosen for a specific function within a microprocessor-based system generally depends upon which features of the memory are best suited to perform the particular function. There is often a tradeoff between speed and cost of memory devices. Memory manufacturers provide an array of innovative, fast memory chips for various applications.

Dynamic Random Access Memory (DRAM) devices are generally used for main memory in computer systems because they are relatively inexpensive. When higher data rates are necessary, Static Random Access Memory (SRAM) devices may be incorporated at a higher cost. To strike a balance between speed and cost, computer systems are often configured with cache memory.

Cache memory is a special high-speed storage mechanism which may be provided as a reserved section of the main memory or as an independent high-speed storage device. A memory cache is a portion of the memory which is made of the high speed SRAM rather than the slower and cheaper DRAM which is used for the remainder of the main memory. Memory caching is effective since most computer systems implement the same programs and request access to the same data or instructions repeatedly. By storing frequently accessed data and instructions in the SRAM, the system can minimize its access to the slower DRAM.

Some memory caches are built into the architecture of the microprocessor themselves, such as the Intel 80486 microprocessor and the Pentium processor. These internal caches are often called level 1 (L1) caches. However, many computer systems also include external cache memory or level 2 (L2) caches. These external caches sit between the central processing unit (CPU) and the DRAM. Thus, the L2 cache is a separate chip residing externally with respect to the microprocessor. However, despite the apparent discontinuity in nomenclature, more and more

microprocessors are incorporating larger caches into their architecture and referring to these internal caches as L2 caches. Regardless of the term used to describe the memory cache, the memory cache is simply an area of memory which is made of Static RAM to facilitate rapid access to often used information.

5

As previously discussed, frequently accessed data may be stored in the cache memory area of main memory. Thus, the portion of the system which is accessing the main memory should be able to identify what area of main memory it must access to retrieve the required information. A “tag RAM” identifies which data from the main memory is currently stored in each cache line. The data is stored in the cache. The values stored in the tag RAM determine whether the actual data can be retrieved quickly from the cache or if the requesting device will have to access the slower DRAM portion of the main memory. The size of the data store determines how much data the cache can hold at any one time. The size of the tag RAM determines what range of main memory can be cached. Many computer systems, for example, are configured with a 256k L2 cache and tag RAM that is 8 bits wide. This is sufficient for caching up to 64 MB of main memory.

10  
15  
20  
25  
30  
35  
40  
45  
50  
55  
60  
65  
70  
75  
80  
85  
90  
95  
100  
105  
110  
115  
120  
125  
130  
135  
140  
145  
150  
155  
160  
165  
170  
175  
180  
185  
190  
195  
200  
205  
210  
215  
220  
225  
230  
235  
240  
245  
250  
255  
260  
265  
270  
275  
280  
285  
290  
295  
300  
305  
310  
315  
320  
325  
330  
335  
340  
345  
350  
355  
360  
365  
370  
375  
380  
385  
390  
395  
400  
405  
410  
415  
420  
425  
430  
435  
440  
445  
450  
455  
460  
465  
470  
475  
480  
485  
490  
495  
500

In a multi-processor system, each processor may have a corresponding main memory, with each main memory reserving a portion for cache memory. The process of managing the caches in a multi-processor system is complex. “Cache coherence” refers to a protocol for managing the caches of a multi-processor system so that no data is lost or over-written before the data is transferred from a cache to a requesting or target memory. Each processor may have its own memory cache that is separate from a larger shared RAM that the individual processors will access. When these multi-processors with separate caches share a common memory, it is necessary to keep

20

the caches in a state of coherence by insuring that any shared operand that has changed in any cache is changed throughout the entire system. Cache coherency is generally maintained through either a directory based or a snooping system. In a directory based system, the data being shared is placed in a common directory that maintains the coherence between the caches. The directory acts as a filter through which the processor must ask permission to load an entry from the primary memory to its cache. When an entry is changed, the directory either updates or invalidates the other caches with that entry. Disadvantageously, directory based coherency systems add to the cycle time (previously reduced by the implementation of cache memory) by requiring that each access to the cache memory go through the common directory. In typical snooping systems, all caches on a bus monitor (or snoop) the bus to determine if they have a copy of the block of data that is requested on the bus. Every cache has a copy of the sharing status of every block of physical memory it has.

Thus, cache coherence aims at solving problems associated with sharing data in a multi-processor computer system which maintains separate caches. This problem is further promulgated when the computer system includes multiple processor buses. In a multi-processor-bus shared memory system, a host controller must maintain memory coherency throughout all the processor caches. This requires snoop cycles to be run on buses other than the originating bus. A snoop filter may be implemented to minimize the amount of snoop cycles on other buses. To maintain efficiency, the snoop filter should facilitate data look up from the tag RAMs as quickly as possible to determine the proper snoop response on the processor bus in a minimal number of clock cycles.

The present invention may be directed to one or more of the problems set forth above.

### **BRIEF DESCRIPTION OF THE DRAWINGS**

The foregoing and other advantages of the invention will become apparent upon reading the following detailed description and upon reference to the drawings in which:

Fig. 1 illustrates a block diagram of an exemplary multi-processor-bus computer system;

Fig. 2 illustrates a block diagram of a snoop filter in accordance with the present technique;

Fig. 3 illustrates a more detailed block diagram of an exemplary embodiment of the snoop filter of Fig. 2; and

Fig. 4 illustrates a block diagram of an exemplary two dimensional doubly linked list structure.

### **DETAILED DESCRIPTION OF SPECIFIC EMBODIMENTS**

One or more specific embodiments of the present invention will be described below. In an effort to provide a concise description of these embodiments, not all features of an actual implementation are described in the specification. It should be appreciated that in the development of any such actual implementation, as in any engineering or design project, numerous implementation-specific decisions must be made to achieve the developers' specific goals, such as compliance with system-related and business-related constraints, which may vary from one implementation to another. Moreover, it should be appreciated that such a

development effort might be complex and time consuming, but would nevertheless be a routine undertaking of design, fabrication, and manufacture for those of ordinary skill having the benefit of this disclosure.

5           Turning now to the drawings and referring initially to Fig. 1, a block diagram of an exemplary multi-processor-bus computer system is illustrated and designated generally as reference numeral 10. The computer system 10 typically includes one or more processors or CPUs. In the exemplary embodiment, the system 10 utilizes eight microprocessors 12A-12H. The system 10 utilizes a split bus configuration in which the processors 12A-12D are coupled to a first bus 14A, whereas the processors 12E-12H are coupled to a second bus 14B. It should be understood that the processor or processors 12A-12H may be of any suitable type, such as a microprocessor available from Intel, AMD, or Motorola, for example. Furthermore, any suitable bus arrangement may be coupled to the processors 12A-12H, such as a split bus (as illustrated), or individual buses. By way of example, the exemplary system 10 may utilize Intel Pentium III processors and the buses 14A and 14B may operate at 100/133 MHz.

Each of the buses 14A and 14B is coupled to a chip set which includes a host controller 16 and a data controller 18. In this embodiment, the data controller 18 is effectively a data cross bar slave device controlled by the host controller 16. Therefore, these chips will be referred to together as the host/data controller 16,18. The host/data controller 16,18 is further coupled to one or more memory controllers. In this particular example, the host/data controller 16,18 is coupled to five memory controllers 20A-20E via five individual bus segments 22A-22E, respectively. Each of the memory controllers 20A-20E is further coupled to a segment of main



memory designated as 24A-24E, respectively. As discussed in detail below, each of the memory segments or modules 24A-24E is typically comprised of dual inline memory modules (DIMMs). Further, each memory module 24A-24E and respective memory controller 20A-20E may comprise a single memory cartridge 25A-25E which may be removable. In the present configuration, data may be stored in a "4+1" parity striping pattern wherein one of the memory cartridges 25A-25E is used to provide redundancy. Collectively, the memory cartridges 25A-25E (containing the memory modules 24A-24E) make up the RAM memory 26 of the system 10. Further, the system 10 includes an area of cache memory, functionally illustrated as tag RAM 29.

The host/data controller 16,18 is typically coupled to one or more bridges 28A-28C via an input/output (I/O) bus 27. The opposite side of each bridge 28A-28C is coupled to a respective bus 30A-30C, and a plurality of peripheral devices 32A and 32B, 34A and 34B, and 36A and 36B may be coupled to the respective buses 30A, 30B, and 30C. The bridges 28A-28C may be any of a variety of suitable types, such as PCI, PCI-X, EISA, AGP, etc.

As previously discussed, each CPU 12A-12H may include a segment of cache memory for storage of frequently accessed data and code. Coherency between the caches found in each CPU 12A-12H is further complicated by the split bus configuration since coherency must be maintained between the separate buses. Also, because requests may originate from or be directed to not only one of the CPUs 12A-12H but also from one of the peripheral devices 32A-B, 34A-B, or 36A-B, cache coherency must be maintained along the I/O bus 27, as well. To maintain coherency, a mechanism is provided in the host/data controller 16, 18 to efficiently facilitate snooping of the buses in the present multi-processor/ multi-bus system 10.

The host controller 16 typically includes a processor controller (PCON) for each of the processor and I/O bus 14A, 14B, and 27. For simplicity, the processor controller corresponding to the processor bus 14A is designated as "PCON0." The processor controller corresponding to the processor bus 14B is designated as "PCON1." The processor controller corresponding to the I/O bus 27 is designated as "PCON2." Essentially, each processor controller PCON0-PCON2 serves the same function which is to connect the buses which are external to the host controller 16 (i.e., processor bus 14A and 14B and I/O bus 27) to the internal blocks of the host controller 16. Thus, PCON0-PCON2 provide the interfaces between the buses 14A, 14B, and 27 and the host controller 16.

Fig. 2 illustrates a block diagram of a tag control (TCON) module 40 which may be used in accordance with the present techniques to manage the snooping process efficiently. That is to say, the TCON module 40 serves as a snoop filter to minimize the number of clock cycles required to obtain cache state information and to determine the proper snoop response on the processor or I/O buses 14A, 14B, and 27. In the present embodiment, the TCON module 40 resides within the host controller 16. Alternatively, the TCON module 40 may reside in some external device which has access to each of the processor buses 14a and 14B and the I/O bus 27.

There are four main functional blocks within the TCON module 40. The first block is the request module 42. The request module 42 accepts the cycle requests from the processor controllers PCON0, PCON1, and PCON2 (Fig. 1) and schedules the cycles to be run through the tag look up. Generally, the request module 42 maintains proper request order, prioritizes the input, and establishes each of the queues or list structures. The request module 42 insures

memory coherency through proper cycle order, arbitrates access to the memory among the processor and I/O buses 14a, 14B, and 27, and optimally utilizes the tag look up bandwidth.

As previously stated, the request module 42 is responsible for maintaining proper cycle order. This is accomplished using content addressable memory (CAM) of new cycles against existing cycles. Generally, the CAM requests provide a means of insuring that memory WRITES are only performed when necessary. Because many READ and WRITE requests are active at any given time on the memory and I/O buses 14A, 14B, and 27, a CAM request is issued to insure that the READ and WRITE requests are performed only when necessary and according to proper dependencies. When one of the processor controllers PCON0-PCON2 issues a new cycle request, the cycle checks for cycle order dependencies across the existing cycles in the request buffer. READs are compared against all WRITES across all buses 14A, 14B and 27 to preserve data coherency. Further, READs are compared against all READs on the same bus to preserve data coherency through proper data order. Only one memory READ can be deferred at a time. If a cycle is deferrable, subsequent cycles are retried. Otherwise, the subsequent cycles are held in a parent/child list which defines the priority of the dependent requests (or children) based on the priority of the primary request (or parent). If, for instance, a WRITE request is issued to a particular location in memory, and subsequently, a READ request is issued to that same location in memory, the request module 42 will insure that the WRITE request is executed before the READ request. When the READ request is submitted, it is compared to all outstanding WRITE requests. Because READ requests are typically prioritized ahead of WRITE requests, without a CAM cycle, the READ could be processed before the WRITE request and thus return old or

invalid data from the memory location. The request module 42 will be further discussed with reference to Fig. 3.

The TCON module 40 also includes an active snoop queue (ASQ) 44 which may include one or more buffers which contain the indices of all requests which are currently active in the TCON module 40. The indices are used to prevent multiple accesses to the same index simultaneously. In general, the ASQ 44 includes buffers to maintain a list of current cycles that are being processed through the tag RAM interface module 46. The ASQ 44 only permits one access per cache line index at a time to maintain proper cycle order and cache state information. In the present embodiment, there is one ASQ 44 for each tag RAM interface module 46. A single ASQ 44 and tag RAM interface module 46 are illustrated in Fig. 2. However, as further explained with reference to Fig. 3, it may be advantageous to incorporate more than one ASQ 44 and tag RAM interface module 46. In the present embodiment, each ASQ 44 permits sixteen cycles to be active at a time, with each of these cycles being processed independently. Because the ASQ 44 buffers include a fixed number of locations, the number of tag lines that are currently active is limited by the size of the buffers in the ASQ 44. The ASQ 44 will be further discussed with reference to Fig. 3.

The tag RAM interface module 46 provides the interface with the tag RAM 29. The tag RAM 29 is a separate memory storage device from main memory 26 that keeps track of all cachelines in use by the CPUs 12A-12H. Thus, the tag RAM 29 provides a look-up table for the cache state in cache memory. The tag RAM interface module 46 is optimized to make efficient use of the bandwidth in the SRAM (i.e., cache memory). To minimize the turn around clocks

required when switching between READ and WRITE requests, the WRITE requests are queued so that they may burst directly to the SRAM in back-to-back clock cycles. The tag RAM interface module 46 also reduces the number of READs to the SRAM whenever possible by interpreting the cache data based on the cycle characteristics. This may occur if a cycle has a non-cacheable attribute or if the cycle is an explicit writeback. If the cycle has a non-cacheable attribute, there is no reason to check the tag RAM since the requested information will not be stored in the cache memory. Conversely, if the request is an explicit writeback (which is a cycle initiated by a CPU 12A-12H to update the memory 26 with modified data), the requested data is known to be in the cache memory, without checking the tag RAM 29. When one of these cycles occur, the tag RAM interface module 46 may switch from a READ mode to a WRITE mode. The tag RAM interface module 46 also minimizes the amount of WRITES to the SRAM whenever the processor performs a WRITE to an unallocated address. This preserves the address that was allocated prior to the WRITE and alleviates the need to perform an unnecessary castout which is a forced expulsion of old data from the cache. Finally, the tag RAM interface module 46 has programmable state transition values that can be tuned to optimize performance based on testing. The programmability includes selecting shared versus owned on a code READ and shared versus owned on a data READ. As can be appreciated by those skilled in the art, basic cache protocol dictates that data be one of the four types: modified, exclusive, shared, or invalid (MESI). "Owned" data refers to data that is either modified or exclusive. The functionality of the tag RAM interface module 46 will be better understood in light of Fig. 3 and the corresponding description.

The response module 50 receives the current tag state information for a cycle being processed by the ASQ 44. Based on the tag state value, castout snoops, shared snoops, or invalidate snoops may be run on one or more of the processor buses 14A and 14B to maintain cache coherency. A castout snoop is run when the tag index of the current cycle does not match the value in the tag RAM 29. A shared snoop is run when a cacheline must be removed from modified or exclusive to shared. An invalidate snoop is run when a cacheline must be removed from modified, exclusive, or shared to invalid. After all required snoops have been completed, the response module 50 updates the tag RAM 29 with the new state information and releases the slot occupied in the ASQ 44.

Turning now to Fig. 3, a more detailed description of the TCON module 40 and the data path through the TCON module 40 is illustrated. As previously discussed, the TCON module 40 receives input from the processor controllers PCON0-PCON2. Specifically, the processor controllers PCON0, PCON1, and PCON2, designated as 52A-52C, each monitor a corresponding bus and sends cycle requests to the appropriate destination (e.g., I/O or memory). Requests from each processor controller 52A-52C are then stored in respective READ and WRITE buffers 54A-54C and 56A-56C until the requests are dispersed to their proper destination. In this embodiment, each READ buffer 54A-54C and each WRITE buffer 56A-56C can store up to sixteen READ and WRITE cycles respectively. A request is delivered from the processor controller 52A-52C to the READ buffer 54A-54C or WRITE buffer 56A-56C along a request path 58A-58C. The request signals which are delivered along the request path 58A-58C may be buffered for fanout purposes. In the present embodiment, the entries into the READ and WRITE buffers 54A-54C and 56A-56C may be indexed using bits 3-0 in the request. Bit 4 may be set to

distinguish between READ requests and WRITE requests and therefore provides the routing information to send the request to the proper buffer 54A-54C or 56A-56C.

As previously discussed, for each request which is stored in a respective READ buffer 54A-54C, the READ request is advantageously compared against WRITE requests across all processor controllers 52A-52C and against all READ requests on the same processor controller 52A-52C to preserve data coherency through a CAM request. The request module 42 controls the CAM requests 62A-62C for each of the respective READ buffers 54A-54C. Depending on the response from the CAM request 62A-62C, the request module 42 may reprioritize the present READ request.

A primary concern in maintaining cache coherency is to find the state of the cache line in the system as efficiently as possible. Memory READs are generally higher priority than memory WRITEs since providing READ requests as quickly as possible is advantageous. WRITE requests can generally be fulfilled with less time criticality. There may be exceptions when, for instance, the WRITE buffers 56A-56C become full or reach a threshold or when READ requests are dependent on a WRITE request. To provide the requested information as quickly as possible, a bypass path 64A-64C is provided for each processor controller 52A-52C. Often, the READ request can be sent immediately to the tag RAM interface module 46 if there are no other requests waiting to be run in the system. Each time a READ request is delivered, it is sent quickly through the bypass path 64A-64C, thus bypassing storage and compare logic which may be unnecessary. On a first clock cycle, the READ request is sent along the bypass path 64A-64C. On the next clock cycle, the CAM request 62A-62C is performed. If there are no other

requests at that particular address, no additional management steps are used to maintain coherency. If the CAM request 62A-62C detects a WRITE request to the same address, the response module sends an abort through the system to cancel the READ request which was sent along the bypass path 64A-64C. The abort command is sent on the next clock cycle.

5 Essentially, the speed of the processing of the READ requests is made more efficient by forcing the READ request quickly through the system and subsequently checking to see if the READ request is going to be valid. If not, then the READ request sent ahead on the bypass path 64A-64C is aborted, and the READ request is then prioritized using the list structures and arbitration mechanisms discussed below. The request signals delivered along the bypass path 64A-64C may  
10 be buffered for fanout purposes.

The TCON module 40 incorporates a plurality of list structures to maintain the order of the requests in the READ and WRITE buffers 54A-54C and 56A-56C. The WRITE list structures 56A-56C are generally first-in-first-out (FIFO) such that requests are processed in the  
15 order in which they were received, thereby fulfilling the ordering requirements for the WRITE requests. This simple processing scheme is generally adequate for the WRITE requests since they are generally less time critical than the READ requests. The READ requests are generally more complicated based on the prioritization and arbitration scheme discussed herein. Generally, the list structures are designed as two dimensional doubly-linked lists. These lists insure that  
20 proper cycle order is maintained for each type of transaction. The design of the lists allow for both the insertion and deletion of a cycle request in a single clock cycle. In addition, the READ request list for each processor controller 52A-52C is designed to immediately rotate a request cycle from the head to the tail of the list whenever the required resources are unavailable, thus



allowing other READ requests to gain access to the resources. The list structures are described in more detail below and discussed further with reference to Fig. 4.

As previously explained, each processor controller 52A-52C connects an external bus, such as the processor bus 14A or 14B or the I/O bus 27, to the internal blocks of the host controller 16. To identify the individual list structures, the nomenclature described in table 1 may be helpful. Table 1 describes requests between the input/output ("I"), the processor ("P"), and the memory ("M"). Thus, transactions involving one of the CPUs 12A-12H will include a P in the name of the list structure, and so forth. Thus, the list structure I2P refers to READ requests from one of the processors 12A-12H to one of the devices on the I/O bus 27. The P2I list structure refers to a WRITE request from one of the processors 12A-12H to one of the devices on the I/O bus 27. M2P-0 refers to the list structure containing READ requests from the memory 26 to one of the processors 12A-12D on the processor bus 14A. P2M-0 refers to the list structure wherein one of the processors 12A-12D on the processor bus 14A wants to WRITE to memory 26. For transactions involving the processors 12E-12H on processor bus 14B, the list structures are designated M2P-1 and P2M-1 referring to the corresponding READ and WRITE requests between the processors 12E-12H on processor bus 14B and the memory 26. The M2I list structure refers to the READ requests from the devices on the I/O bus 27 to the memory 26. The I2M list structure refers to WRITE requests from devices on the I/O bus 27 to the memory 26. The nomenclature is summarized in Table 1, below.

**Table 1: List Structure Nomenclature**

List Name	Request Type	Description
<b>I2P</b>	<b>READ</b>	<b>I/O (PCON2) wants to read from CPU</b>
<b>P2I</b>	<b>WRITE</b>	<b>I/O (PCON2) wants to write to CPU</b>
<b>M2P-0</b>	<b>READ</b>	<b>CPU (PCON0) wants to read from memory</b>
<b>P2M-0</b>	<b>WRITE</b>	<b>CPU (PCON0) wants to write to memory</b>
<b>M2P-1</b>	<b>READ</b>	<b>CPU (PCON1) wants to read from memory</b>
<b>P2M-1</b>	<b>WRITE</b>	<b>CPU (PCON1) wants to write to memory</b>
<b>M2I</b>	<b>READ</b>	<b>I/O device (PCON2) wants to read from memory</b>
<b>I2M</b>	<b>WRITE</b>	<b>I/O device (PCON2) wants to write to memory</b>

The list structures may contain any number of desirable locations. In the present embodiment, the P2M and M2P list structures contain sixteen locations per PCON0 and PCON1. The I2M and the M2I list structures each contain sixteen locations for PCON2. The P2I and I2P list structures each contain 32 locations, sixteen for PCON0 and sixteen for PCON1.

Dependency pointers are used to guarantee proper ordering of cycles to preserve the memory coherency and proper request processing order. The dependencies are established at the time a new request is delivered to a READ or WRITE buffer 54A-54C or 56A-56C. The dependency status is cleared whenever a cycle is removed from the READ or WRITE buffers 54A-54C and 56A-56C. A CAM request 62A-62C is performed using the ID of the retired cycle against all dependencies. When a new memory READ request is received (i.e., M2P or M2I), a

CAM lookup of the address is performed against the WRITE request buffer of all three PCONs 52A-52C. If a "CAMHIT" results, a dependency of the READ to follow the last WRITE for each PCON 52A-52C with a CAMHIT is established. Further, when a new I2P request is received, a dependency of the READ to follow the last cycle, if any, in the P2I list is structure established. Finally, when a new P2I request is received, a dependency of the WRITE to follow the last cycle, if any, in the I2P list structure is established. When a cycle is retired out of order due to a CAMHIT, all dependencies on that cycle are modified to point to the previous cycle in the corresponding list. If there is no previous cycle, the dependency status is cleared.

The "head" and "tail" pointer structures associated with each list structure indicate the first and last locations in the list structure which are filled. When a request is added to the list structure, it is submitted to the tail of the list structure. Once a request reaches the head of the list structure, the request will be processed unless it has a dependency in which case the request will be moved to the tail of the list structure.

There are two types of list structures utilized to maintain order of the cycles stored in the READ request buffers 54A-54C: Active List and Parent Child List. The WRITE request buffers 56A-56C utilize only the Active List. The Active List contains pointers to cycles that are available for arbitration to the ASQ 44. The structure is a double linked list with a head and tail pointer as previously discussed and further discussed with reference to Fig. 4, below. Each cycle in the request buffers 54A-54C and 56A-56C corresponds to a location in a next-pointer array and the previous-pointer array. READ and WRITE cycles are never combined into the same list structure so the pointers do not need to maintain the READ/WRITE bit from the cycle request id

(bit 4 in the example described above). The Parent Child List contains pointers to memory READ cycles that are to the same address as another memory READ cycle active in the request buffer within the same PCON 52A-52C. The structure is a double-linked list without a head or tail pointer as will be discussed further with reference to Fig. 4 below. If a request cycle does not have a parent, it appears on the Active List. Conversely, if an active cycle does have a parent, it appear on the Parent Child List. A request cycle is assigned to the Parent Child List if a READ request is dependent on another READ request occurring first. It will be doubly-linked below the parent on which it depends. Once the parent request is processed, the child request is moved to the tail end of the Active List.

To reduce the latency in an idle system, requests to the tag RAM interface 46 proceed immediately when a request is received. One clock cycle later, the status of the ASQ CAMHIT is valid, indicating whether the requested address was found. If true, the abort status of the cycle will be set. This results in a slot assigned to the request in the active snoop buffer being marked available. Further, upon completion of the tag lookup from SRAM, if the cycle was not aborted, the tag RAM interface module 46 will transfer the results to the TCON response module 50.

The cycle request arbitration portion of the Request module 42 is designed to optimize the utilization of the two tag RAM interface 46 and includes a plurality of multiplexors. The present embodiment illustrates a system wherein the even and odd addresses of each request have been separated to be processed by individual portions of the TCON module 40. Thus, the TCON module 40 illustrated in Fig. 3 illustrates even and odd arbitration logic (multiplexors, discussed

below), even and odd ASQs (active snoop buffer) 44A and 44B and even and odd tag RAM interface modules 46A and 46B. In an alternate embodiment, a different arbitration configuration and a single ASQ 44 and tag RAM interface module 46 may be implemented. Further, in another embodiment, it may be advantageous to implement more than two ASQs 44A and 44B and more than two tag RAM interfaces 46A and 46B. The arbiters efficiently assign requests for the READ queue and WRITE queue of each of the three processor controllers 52A-52C (PCON0, PCON1, and PCON2) to the tag RAM interfaces 46A and 46B. READ requests from one of the three processor controllers 52A-52C bypass the front end list structures and are sent immediately to one of the tag RAM interfaces 46A and 46B via the bypass path 64A-64C prior to performing any qualifying checks on the READ. On the next clock, the qualifying checks are performed. If a conflict is detected, the READ request to the tag RAM interface 46A or 46B is aborted.

READ requests that could not bypass the front and list structures are normally arbitrated to the tag RAM interface 44A or 44B at higher priority than WRITES since WRITES are posted and do not affect system performance. As previously discussed, WRITES are dynamically arbitrated at higher priority than READs under two conditions: 1) there is a READ cycle that has a dependency upon a WRITE cycle; and 2) the number of queued WRITES has surpassed some programmable threshold. The second condition may be set by the BIOS or firmware to minimize the system delays as a result of a backlog in the WRITE queue. In certain systems, it may be advantageous, for instance, to flush a WRITE queue once there are twelve requests sitting in the WRITE buffer. In this case the WRITE buffer will be flushed and the WRITE requests will be processed while the READ requests, which are normally prioritized higher than the WRITE

requests, are stalled until either the WRITE buffer is emptied or the requests in the WRITE buffer are reduced to some minimum threshold value, such as four.

Cycle disbursement may be facilitated as follows. For the P2I and the I2P cycle requests (i.e., the WRITE and READ requests between PCON2 from the I/O bus and the CPUs 12A-H) are dispersed to the I/O queue in the order that the cycles are received. This is illustrated in Fig. 3 by arbiters 70 and 72 which may be multiplexors, for example. The arbitration scheme for requests to main memory 26 is more complicated. For memory access arbitrations, there are two levels of arbitration for both the even and odd active buffers: "intra-PCON" and "inter-PCON." Generally, a first multiplexor 74 is provided for each of the remaining list structures, as with the I2P and P2I structures 70 and 72. The multiplexor 74 cycles the requests through from the corresponding READ or WRITE buffer 54A-54C or 56A-56C in the order in which they are received. Next, the requests are divided based on their even or odd address. Thus, even and odd arbitration occurs independently. The next arbitration mechanism, here a multiplexor 76, arbitrates between all requests from one particular processor controller (PCON0, PCON1, or PCON2) 52A-52C. The arbitration performed by the multiplexor 76 for each processor controller is referred to as intra-PCON arbitration. The intra-PCON arbitration has a four level priority arbitration: high, medium, low, and none. High priority may be assigned to those WRITE requests which have a READ dependency thereon. Medium priority may be assigned to all other READ requests and to WRITE requests when a threshold value is exceeded in the list structures, as previously discussed. Low priority may be assigned to all other WRITE requests without extenuating circumstances. No priority is assigned if the request queue is empty, for example.

Once the intra-PCON arbitration is settled for a particular PCON, the multiplexor 76 will output the highest priority request on that particular PCON to an inter-PCON arbitration mechanism, here a multiplexor 78. The inter-PCON arbitration executes a round robin arbitration between the three requesting agents (PCON0, PCON1, and PCON2). Each multiplexor 78 checks each request from PCON0, PCON1, and PCON2 for a high priority request. If one of the inputs includes a high priority request, it will be processed first. Then the next input is checked. For example, the multiplexor 78 may first check the input from PCON0. If the request has a high priority, it will be processed through the multiplexor first. If it does not, the multiplexor 78 will check the request from PCON1. If the request on PCON1 was of high priority, it will be processed through the multiplexor 78. If this request is not high priority the multiplexor 78 will check PCON2 to check if there is a high priority on the input of the multiplexor 78. Again, if there is a high priority, the request will be processed through the multiplexor 78. The process is then repeated starting again at PCON0, going through the medium requests of each of the PCON channels, and then repeating again for the low priority requests. As each request is processed through the multiplexor 78, it is sent to a respective ASQ 44A or 44B. The ASQ cycle tracker maintains a list of current cycles that are being processed through the tag RAM interface 46A or 46B as previously discussed.

Fig. 4 illustrates an exemplary two dimensional, doubly-linked list structure 90 (e.g. M2P-0, P2M-0, M2I, etc.), as previously described with reference to Fig. 3. The list structure 90 includes the proper ordering for the processing of each of the requests in the respective read and write buffers 54A-54C and 56A-56C. Each list structure 90 includes a head pointer 92 and a tail pointer 94 to insure that proper cycle order is maintained. The head and tail pointers 92 and 94

associated with each list structure 90 indicate the first and last locations in the list structure 90 which are filled with requests (READ or WRITE, depending on the particular list structure 90). Thus, when a request is linked (or pointing to) the head pointer 92, as with request A in Fig. 4, that request is the next request to be processed. When a request is added to the list structure 90, it is submitted to the tail of the list structure. Once a request reaches the head of the list structure, the request will be processed unless it has a dependency in which case the request will be moved to the tail of the list structure, as will be described below.

The list structure 90 includes an Active List 96 and a Parent Child List 98. The Active List 96 contains pointers to cycles that are available for arbitration to the ASQ 44. The Active List 96 of the list structure 90 is a double linked list with a head and tail pointer 92 and 94. Each cycle in the request buffer corresponds to a location in a next-pointer array and the previous-pointer array. For instance, Request A is linked to the Head Pointer 92, as well as Request B. On a first transition, Request A will be processed and the link from Request B to Request A will be replaced with a link from Request B to the head pointer 92. On the next transition, Request B will be processed since it is now linked to the head pointer 92. The design of the list structure 90 allows for both the insertion and deletion of a cycle request in a single clock cycle. Thus, if for instance, Request B is cancelled, it will be removed from the list structure 90, and on the same transition, request C will be linked to Request A. In addition, for a list structure associated with a READ request the list is designed to immediately rotate a request cycle from the head to the tail of the list whenever the required resources are unavailable, thus allowing other READ requests to gain access to the resources.



The Parent Child List 98 of the list structure 90 manages the dependencies associated with particular requests with need to be performed in a particular order. A request cycle is assigned to the Parent Child List 98, if the request is to the same address as another request occurring first and it will be doubly linked below its parent on which it depends. Dependency pointers are used to guarantee proper ordering of cycles to preserve the memory coherency and proper program order. The dependencies are established through the CAM requests 62A-62C at the time a new request is delivered to a READ or WRITE buffer 54A-54C or 56A-56C, as previously discussed. The dependency status is cleared whenever a cycle is removed from the READ or WRITE buffers 54A-54C and 56A-56C. The Parent Child List 98 contains pointers to memory READ cycles that are to the same address as another memory READ cycle active in the request buffer within the same processor controller (PCON0-2) 52A-52C. The structure is a double linked list without a head or tail pointer. In the exemplary list structure illustrated in Fig. 4, Request D is dependent on Request C, and Request E is dependent on Request D. Each request is doubly linked and thus points to the proceeding and subsequent requests. As with the Active List 96, when a request is retired out of order due to a CAMHIT, for example, all dependencies on that cycle are modified to point to the previous cycle in the corresponding list. Thus, if Request D is cancelled, Request E will be linked to Request C. If there is no previous cycle, the dependency status is cleared.

If a Request does not have a parent, as with Requests A, B, and C, it appears on the Active List 96. Conversely, if a Request does have a parent, as with Requests D and E, it appears on the Parent Child List 98. Once the parent request is processed, the child request will

be moved to the tail end of the Active List 96. Thus, in the present example, once Request C is processed, Request D will be moved to the end of the Active List 96.

While the invention may be susceptible to various modifications and alternative forms,  
5 specific embodiments have been shown by way of example in the drawings and will be described  
in detail herein. However, it should be understood that the invention is not intended to be limited  
to the particular forms disclosed. Rather, the invention is to cover all modifications, equivalents  
and alternatives falling within the spirit and scope of the invention as defined by the following  
appended claims.

10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241  
1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295  
1296  
1297  
1298  
1299  
1300  
1301  
1302  
1303  
1304  
1305  
1306  
1307  
1308  
1309  
1310  
1311  
1312  
1313  
1314  
1315  
1316  
1317  
1318  
1319  
1320  
1321  
1322  
1323  
1324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349  
1350  
1351  
1352  
1353  
1354  
1355  
1356  
1357  
1358  
1359  
1360  
1361  
1362  
1363  
1364  
1365  
1366  
1367  
1368  
1369  
1370  
1371  
1372  
1373  
1374  
1375  
1376  
1377  
1378  
1379  
1380  
1381  
1382  
1383  
1384  
1385  
1386  
1387  
1388  
1389  
1390  
1391  
1392  
1393  
1394  
1395  
1396  
1397  
1398  
1399  
1400  
1401  
1402  
1403  
1404  
1405  
1406  
1407  
1408  
1409  
1410  
1411  
1412  
1413  
1414  
1415  
1416  
1417  
1418  
1419  
1420  
1421  
1422  
1423  
1424  
1425  
1426  
1427  
1428  
1429  
1430  
1431  
1432  
1433  
1434  
1435  
1436  
1437  
1438  
1439  
1440  
1441  
1442  
1443  
1444  
1445  
1446  
1447  
1448  
1449  
1450  
1451  
1452  
1453  
1454  
1455  
1456  
1457  
1458  
1459  
1460  
1461  
1462  
1463  
1464  
1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495  
1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511  
1512  
1513  
1514  
1515  
1516  
1517  
1518  
1519  
1520  
1521  
1522  
1523  
1524  
1525  
1526  
1527  
1528  
1529  
1530  
1531  
1532  
1533  
1534  
1535  
1536  
1537  
1538  
1539  
1540  
1541  
1542  
1543  
1544  
1545  
1546  
1547  
1548  
1549  
1550  
1551  
1552  
1553  
1554  
1555  
1556  
1557  
1558  
1559  
1560  
1561  
1562  
1563  
1564  
1565  
1566  
1567  
1568  
1569  
1570  
1571  
1572  
1573  
1574  
1575  
1576  
1577  
1578  
1579  
1580  
1581  
1582  
1583  
1584  
1585  
1586  
1587  
1588  
1589  
1590  
1591  
1592  
1593  
1594  
1595  
1596  
1597  
1598  
1599  
1600  
1601  
1602  
1603  
1604  
1605  
1606  
1607  
1608  
1609  
1610  
1611  
1612  
1613  
1614  
1615  
1616  
1617  
1618  
1619  
1620  
1621  
1622  
1623  
1624  
1625  
1626  
1627  
1628  
1629  
1630  
1631  
1632  
1633  
1634  
1635  
1636  
1637  
1638  
1639  
1640  
1641  
1642  
1643  
1644  
1645  
1646  
1647  
1648  
1649  
1650  
1651  
1652  
1653  
1654  
1655  
1656  
1657  
1658  
1659  
1660  
1661  
1662  
1663  
1664  
1665  
1666  
1667  
1668  
1669  
1670  
1671  
1672  
1673  
1674  
1675  
1676  
1677  
1678  
1679  
1680  
1681  
1682  
1683  
1684  
1685  
1686  
1687  
1688  
1689  
1690  
1691  
1692  
1693  
1694  
1695  
1696  
1697  
1698  
1699  
1700  
1701  
1702  
1703  
1704  
1705  
1706  
1707  
1708  
1709  
1710  
1711  
1712  
1713  
1714  
1715  
1716  
1717  
1718  
1719  
1720  
1721  
1722  
1723  
1724  
1725  
1726  
1727  
1728  
1729  
1730  
1731  
1732  
1733  
1734  
1735  
1736  
1737  
1738  
1739  
1740  
1741  
1742  
1743  
1744  
1745  
1746  
1747  
1748  
1749  
1750  
1751  
1752  
1753  
1754  
1755  
1756  
1757  
1758  
1759  
1760  
1761  
1762  
1763  
1764  
1765  
1766  
1767  
1768  
1769  
1770  
1771  
1772  
1773  
1774  
1775  
1776  
1777  
1778  
1779  
1780  
1781  
1782  
1783  
1784  
1785  
1786  
1787  
1788  
1789  
1790  
1791  
1792  
1793  
1794  
1795  
1796  
1797  
1798  
1799  
1800  
1801  
1802  
1803  
1804  
1805  
1806  
1807  
1808  
1809  
1810  
1811  
1812  
1813  
1814  
1815  
1816  
1817  
1818  
1819  
1820  
1821  
1822  
1823  
1824  
1825  
1826  
1827  
1828  
1829  
1830  
1831  
1832  
1833  
1834  
1835  
1836  
1837  
1838  
1839  
1840  
1841  
1842  
1843  
1844  
1845  
1846  
1847  
1848  
1849  
1850  
1851  
1852  
1853  
1854  
1855  
1856  
1857  
1858  
1859  
1860  
1861  
1862  
1863  
1864  
1865  
1866  
1867  
1868  
1869  
1870  
1871  
1872  
1873  
1874  
1875  
1876  
1877  
1878  
1879  
1880  
1881  
1882  
1883  
1884  
1885  
1886  
1887  
1888  
1889  
1890  
1891  
1892  
1893  
1894  
1895  
1896  
1897  
1898  
1899  
1900  
1901  
1902  
1903  
1904  
1905  
1906  
1907  
1908  
1909  
1910  
1911  
1912  
1913  
1914  
1915  
1916  
1917  
1918  
1919  
1920  
1921  
1922  
1923  
1924  
1925  
1926  
1927  
1928  
1929  
1930  
1931  
1932  
1933  
1934  
1935  
1936  
1937  
1938  
1939  
1940  
1941  
1942  
1943  
1944  
1945  
1946  
1947  
1948  
1949  
1950  
1951  
1952  
1953  
1954  
1955  
1956  
1957  
1958  
1959  
1960  
1961  
1962  
1963  
1964  
1965  
1966  
1967  
1968  
1969  
1970  
1971  
1972  
1973  
1974  
1975  
1976  
1977  
1978  
1979  
1980  
1981  
1982  
1983  
1984  
1985  
1986  
1987  
1988  
1989  
1990  
1991  
1992  
1993  
1994  
1995  
1996  
1997  
1998  
1999  
2000  
2001  
2002  
2003  
2004  
2005  
2006  
2007  
2008  
2009  
2010  
2011  
2012  
2013  
2014  
2015  
2016  
2017  
2018  
2019  
2020  
2021  
2022  
2023  
2024  
2025  
2026  
2027  
2028  
2029  
2030  
2031  
2032  
2033  
2034  
2035  
2036  
2037  
2038  
2039  
2040  
2041  
2042  
2043  
2044  
2045  
2046  
2047  
2048  
2049  
2050  
2051  
2052  
2053  
2054  
2055  
2056  
2057  
2058  
2059  
2060  
2061  
2062  
2063  
2064  
2065  
2066  
2067  
2068  
2069  
2070  
2071  
2072  
2073  
2074  
2075  
2076  
2077  
2078  
2079  
2080  
2081  
2082  
2083  
2084  
2085  
2086  
2087  
2088  
2089  
2090  
2091  
2092  
2093  
2094  
2095  
2096  
2097  
2098  
2099  
2100  
2101  
2102  
2103  
2104  
2105  
2106  
2107  
2108  
2109  
2110  
2111  
2112  
2113  
2114  
2115  
2116  
2117  
2118  
2119  
2120  
2121  
2122  
2123  
2124  
2125  
2126  
2127  
2128  
2129  
2130  
2131  
2132  
2133  
2134  
2135  
2136  
2137  
2138  
2139  
2140  
2141  
2142  
2143  
2144  
2145  
2146  
2147  
2148  
2149  
2150  
2151  
2152  
2153  
2154  
2155  
2156  
2157  
2158  
2159  
2160  
2161  
2162  
2163  
2164  
2165  
2166  
2167  
2168  
2169  
2170  
2171  
2172  
2173  
2174  
2175  
2176  
2177  
2178  
2179  
2180  
2181  
2182  
2183  
2184  
2185  
2186  
2187  
2188  
2189  
2190  
2191  
2192  
2193  
2194  
2195  
2196  
2197  
2198  
2199  
2200  
2201  
2202  
2203  
2204  
2205  
2206  
2207